Package ‘fedmatch’

November 22, 2021

Title Fast, Flexible, and User-Friendly Record Linkage Methods

Version 2.0.5

Description Provides a flexible set of tools for matching two un-linked data sets. ‘fedmatch’ allows for three ways to match data: exact matches, fuzzy matches, and multi-variable matches. It also allows an easy combination of these three matches via the tier matching function.

Depends R (>= 3.5.3)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports stringdist, SnowballC, stringr, purrr, Rcpp, parallel,forcats, data.table, magrittr, scales

RoxygenNote 7.1.2

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

LinkingTo Rcpp, BH

NeedsCompilation yes

Author Melanie Friedrichs [aut], Chris Webster [aut, cre], Blake Marsh [aut], Jacob Dice [aut], Seung Lee [aut]

Maintainer Chris Webster <chris0webster@gmail.com>

Repository CRAN

Date/Publication 2021-11-22 22:10:02 UTC

R topics documented:

articles .......................................................... 2
build_clean_settings ........................................ 3
articles

Data.frame with common articles

Usage

articles

Format

An object of class data.table (inherits from data.frame) with 23 rows and 2 columns.

See Also

clean_strings
**build_clean_settings**  
*Building settings for string cleaning*

**Description**

`build_clean_settings` is a convenient way to make the proper list for the `clean_settings` argument of `tier_match`.

**Usage**

```r
build_clean_settings(
  sp_char_words = fedmatch::sp_char_words,
  common_words = NULL,
  remove_char = NULL,
  remove_words = FALSE,
  stem = FALSE
)
```

**Arguments**

- `sp_char_words` character vector. Data.frame where first column is special characters and second column is full words. The default is
- `common_words` data.frame. Data.frame where first column is abbreviations and second column is full words.
- `remove_char` character vector. String of specific characters (for example, "letters") to be removed
- `remove_words` logical. If TRUE, removes all abbreviations and replacement words in `common_words`
- `stem` logical. If TRUE, words are stemmed

**Value**

list with settings to pass to `clean_strings`

---

**build_corpus**  
*Calculate word corpus for weighted Jaccard matching*

**Description**

Calculate word corpus for weighted Jaccard matching

**Usage**

```r
build_corpus(namelist1, namelist2)
```
Arguments

- `namelist1`: character vector of names from dataset 1
- `namelist2`: character vector of names from dataset 2

Value

A data.table with columns for frequency, inverse frequency, and log inverse frequency for each word in the two strings.

### build_fuzzy_settings

**Build settings for fuzzy matching**

Description

`build_fuzzy_settings` is a convenient way to build the list for the fuzzy settings argument in `merge_plus`.

Usage

```r
build_fuzzy_settings(
  method = "jw",
  p = 0.1,
  maxDist = 0.05,
  matchNA = FALSE,
  nthread = getOption("sd_num_thread")
)
```

Arguments

- `method`: character vector of length 1. Either one of the methods listed in `stringdist::amatch`, or our custom method ‘wgt_jaccard.’ See the vignettes for more details.
- `p`: numeric vector of length 1. See `stringdist::amatch()`
- `maxDist`: numeric vector of length 1. See `stringdist::amatch()`
- `matchNA`: whether or not to match on NAs, see `stringdist::amatch()`
- `nthread`: number of threads to use in the underlying C code.

Value

A list containing options for the 'fuzzy_settings' argument of `merge_plus`. 
build_multivar_settings

Build settings for multivar matching

Description

build_multivar_settings is a convenient way to build the list for the multivar settings argument in merge_plus

Usage

build_multivar_settings(
  logit = NULL,
  missing = FALSE,
  wgts = NULL,
  compare_type = "diff",
  blocks = NULL,
  blocks.x = NULL,
  blocks.y = NULL,
  top = 1,
  threshold = NULL,
  nthread = 1
)

Arguments

- **logit**: a glm or lm model as a result from a logit regression on a verified dataset. See details.
- **missing**: boolean T/F, whether or not to treat missing (NA) observations as its own binary column for each column in by. See details.
- **wgts**: rather than a lm model, you can supply weights to calculate matchscore. Can be weights from calculate_weights.
- **compare_type**: a vector with the same length as "by" that describes how to compare the variables. Options are "in", "indicator", "substr", "difference", "ratio", and "stringdist". See X for details.
- **blocks**: variable present in both data sets to "block" on before computing scores. Match-scores will only be computed for observations that share a block. See details.
- **blocks.x**: name of blocking variables in x. cannot supply both blocks and blocks.x
- **blocks.y**: name of blocking variables in y. cannot supply both blocks and blocks.y
- **top**: integer. Number of matches to return for each observation.
- **threshold**: numeric. Minimum score for a match to be included in the result.
- **nthread**: integer. Number of cores to use when computing all combinations. See parallel::makecluster()

Value

a list containing options for the 'multivar_settings' argument of merge_plus.
build_score_settings Build settings for scoring

Description

build_score_settings is a convenient way to make the proper list for the score_settings argument of merge_plus. Each vector in build_score_settings should be the same length, and each position (first, second, third, etc.) corresponds to one variable to score on.

Usage

build_score_settings(
  score_var_x = NULL,
  score_var_y = NULL,
  score_var_both = NULL,
  wgts = NULL,
  score_type
)

Arguments

- score_var_x: character vector. The variables from the 'x' dataset to score on.
- score_var_y: character vector. The variables from the 'y' dataset to score on.
- score_var_both: the variables from both datasets (shared names) to score on, before any prefixes are applied.
- wgts: numeric vector. The weights for the linear sum of scores.
- score_type: character vector.

Value

A list containing options for the 'score_settings' argument of merge_plus.

build_tier Build settings for a tier

Description

build_tier_settings is a convenient way to make the proper list for the tier_list argument of tier_match. Each vector in build_score_settings should be the same length, and each position (first, second, third, etc.) corresponds to one variable to score on.
Usage

```r
build_tier(
  by.x = NULL,
  by.y = NULL,
  check_merge = NULL,
  match_type = NULL,
  fuzzy_settings = build_fuzzy_settings(),
  score_settings = NULL,
  filter = NULL,
  filter.args = NULL,
  evaluate = NULL,
  evaluate.args = NULL,
  clean_settings = build_clean_settings(),
  clean = NULL,
  sequential_words = NULL,
  allow.cartesian = FALSE,
  multivar_settings = build_multivar_settings()
)
```

Arguments

- **by.x** character string. Variable to merge on in data1. See `merge`
- **by.y** character string. Variable to merge on in data2. See `merge`
- **check_merge** logical. Checks that your unique_keys are indeed unique.
- **match_type** string. If 'exact', match is exact, if 'fuzzy', match is fuzzy. If 'multivar,' match is multivar-based. See `multivar_match`
- **fuzzy_settings** additional arguments for `amatch`, to be used if match_type = 'fuzzy'. Suggested defaults provided. (see `amatch`, method='jw')
- **score_settings** list. Score settings for post-hoc matchscores.
- **filter** function or numeric. Filters a merged data1-data2 dataset. If a function, should take in a data.frame (data1 and data2 merged by name1 and name2) and spit out a trimmed version of the data.frame (fewer rows). Think of this function as applying other conditions to matches, other than a match by name. The first argument of filter should be the data.frame. If numeric, will drop all observations with a matchscore lower than or equal to filter.
- **filter.args** list. Arguments passed to filter, if a function
- **evaluate** Function to evaluate merge_plus output.
- **evaluate.args** list. Arguments passed to evaluate
- **clean_settings** list. Settings for string cleaning. See `clean_strings` and `build_clean_settings`.
- **clean** Boolean, T/F, whether or not to clean strings prior to the match.
- **sequential_words** data.table of words in the same format of the common_words argument in `clean_strings`. Each of these will be replaced from the by columns.
- **allow.cartesian** whether or not to allow many-many matches, see `data.table::merge()`
multivar_settings

list of settings to go to the multivar match if match_type == 'multivar'. See multivar-match.

Value

a list containing 1 tier for the 'tier_list' argument of tier_match.

calculate_weights

Calculate weights for computing matchscore

Description

Calculate weights for comparison variables based on $m$ and $u$ probabilities estimated from a verified dataset.

Usage

calculate_weights(
  data,
  variables,
  compare_type = "stringdist",
  suffixes = c("_1", ",_2"),
  non_negative = FALSE
)

Arguments

data data.frame. Verified data. Should have all of the variables you want to calculate weights for from both datasets, named the same with data-specific suffixes.

variables character vector of the variable names of the variables you want to calculate weights for.

compare_type character vector. One of 'stringdist' (for string variables) 'ratio', 'difference' (for numerics) 'indicator' (0-1 dummy indicating if the two are the same), 'in' (0-1 dummy indicating if data1 is IN data2), and 'substr' (numeric indicating how many digits are the same.)

suffixes character vector. Suffixes of of the variables that indicate what data they are from. Default is same as the default for base R merge, c("x", "y")

non_negative logical. Do you want to allow negative weights?

Details

This function uses the classic Record Linkage methodology first developed by Felligi and Sunter. See Record Linkage. $m$ is the probability of a given link between observations is a true match, while
$u$ is the probability of an unlinked pair of observations being a true match. `calculate_weights` computes a preliminary weight for each variable by computing

$$w = \log_2 \left( \frac{m}{u} \right),$$

then making these weights sum to 1. Thus, the weights that have higher $m$ and lower $u$ probabilities will get higher weights, which makes sense given the definitions. These weights can then be easily passed into the `score_settings` argument of `merge_plus` or `tier_match`, or into the `wgts` argument of `multivar_match`.

**Value**

list with $m$ probabilities, $u$ probabilities, $w$ weights, and settings, the list argument required as an input for `score_settings` in `merge_plus` using the calculate weights.

---

### clean_strings

**String cleaning for easier matching**

**Description**

clean_strings takes a string vector and cleans it according to user-given options.

**Usage**

```r
clean_strings(
  string,
  sp_char_words = fedmatch::sp_char_words,
  common_words = NULL,
  remove_char = NULL,
  remove_words = FALSE,
  stem = FALSE
)
```

**Arguments**

- **string** character or character vector of strings
- **sp_char_words** character vector. Data.frame where first column is special characters and second column is full words. The default is
- **common_words** data.frame. Data.frame where first column is abbreviations and second column is full words.
- **remove_char** character vector. string of specific characters (for example, "letters") to be removed
- **remove_words** logical. If TRUE, removes all abbreviations and replacement words in `common_words`
- **stem** logical. If TRUE, words are stemmed
Details

This function takes a variety of options, each of which changes the behavior. Without the default settings, `clean_strings` will do the following: make the string lowercase; replace special characters &, $, \ names ("and","dollar","percent","at"); convert tabs to spaces and removes extra spaces. This default cleaning puts the strings in a standard format to allow for easier matching.

The other options allow for the removal or replacement of other words or characters.

Value

cleaned strings

corporate_words
corporate_words

Description

Data.frame with common corporate abbreviations in column 1 and corresponding long names in column 2. Useful for cleaning company names for matching.

Usage

corporate_words

Format

An object of class `data.table` (inherits from `data.frame`) with 54 rows and 2 columns.

See Also

clean_strings

Corp_data1

corp_data1
corp_data1

description

Some made up data on the top 10 US companies in the Fortune 500. Mock-matched to `corp_data2` in examples/match_template.R

Usage

corp_data1

Format

An object of class `data.table` (inherits from `data.frame`) with 10 rows and 6 columns.
Description

Some made up data on the top 10 US companies in the Fortune 500. Mock-matched to corp_data1 in examples/match_template.R

Usage

corp_data2

Format

An object of class data.table (inherits from data.frame) with 10 rows and 6 columns.

Description

Data.frame with abbreviations common in the names of financial (i.e. mutual) funds in column 1 and corresponding long names in column 2. Useful for cleaning fund names for matching.

Usage

fund_words

Format

An object of class data.frame with 63 rows and 2 columns.

See Also

clean_strings
fuzzy_match

Use string distances to match on names

**Description**

Use the stringdist package to perform a fuzzy match on two datasets.

**Usage**

```r
def fuzzy_match(
  data1,
  data2,
  by = NULL,
  by.x = NULL,
  by.y = NULL,
  suffixes,
  unique_key_1,
  unique_key_2,
  fuzzy_settings = list(method = "jw", p = 0.1, maxDist = 0.05, matchNA = FALSE,
                nthread = getOption("sd_num_thread"))
)
```

**Arguments**

- `data1` data.frame. First to-merge dataset.
- `data2` data.frame. Second to-merge dataset.
- `by` character string. Variables to merge on (common across data 1 and data 2). See `merge`
- `by.x` character string. Variable to merge on in data1. See `merge`
- `by.y` character string. Variable to merge on in data2. See `merge`
- `suffixes` character vector with length==2. Suffix to add to like named variables after the merge. See `merge`
- `unique_key_1` character vector. Primary key of data1 that uniquely identifies each row (can be multiple fields)
- `unique_key_2` character vector. Primary key of data2 that uniquely identifies each row (can be multiple fields)
- `fuzzy_settings` list of arguments to pass to the fuzzy matching function. See `amatch`

**Details**

stringdist amatch computes string distances between every pair of strings in two vectors, then picks the closest string pair for each observation in the dataset. This is used by fuzzy_match to perform a string distance-based match between two datasets. This process can take quite a long time, for quicker matches try adjusting the nthread argument in fuzzy_settings. The default fuzzy_settings are sensible starting points for company name matching, but adjusting these can greatly change how the match performs.
**Value**

a data.table, the resultant merged data set, including all columns from both data sets.

**Description**

match_evaluate takes in matches and outputs summary statistics for those matches, including the number of matches in each tier and the percent matched from each dataset.

**Usage**

```r
match_evaluate(
matches,
data1,
data2,
unique_key_1,
unique_key_2,
suffixes = c("_1", "_1"),
tier = "tier",
tier_order = NULL,
quality_vars = NULL
)
```

**Arguments**

- `matches` data.frame. Merged dataset.
- `data1` data.frame. First to-merge dataset.
- `data2` data.frame. Second to-merge dataset.
- `unique_key_1` character vector. Primary key of data1 that uniquely identifies each row (can be multiple fields)
- `unique_key_2` character vector. Primary key of data2 that uniquely identifies each row (can be multiple fields)
- `suffixes` character vector. Mnemonics associated data1 and data2.
- `tier` character vector. Default=NULL. The variable that defines a tier.
- `tier_order` character vector. Default= "tier". Variable that defines the order of tiers, if needed.
- `quality_vars` character vector. Variables you want to use to calculate the quality of each tier. Calculates mean.
Details

The most straightforward way to use `match_evaluate` is to pass it to the `evaluate` argument of `tier_match` or `merge_plus`. This will have `merge_plus` return a data.table with the evaluation information, alongside the matches themselves.

`match_evaluate` returns the number of matches in each tier, the number of unique matches in each tier, and the percent matched for each dataset. If no tiers are supplied, the entire dataset will be used as one "tier." The argument `quality_vars` allows for the calculation of averages of any columns in the dataset, by tier. The most straightforward case would be a matchscore, which can again all be done in `merge_plus` with the scoring argument. This lets you see the average matchscore by tier.

Value

data.table. Table describing each tier according to `aggregate_by` variables and `quality_vars` variables.

See Also

`merge_plus`

merge_plus

merge_plus is a wrapper for a standard merge, a fuzzy string match, and a a "multivar" match based on several columns of the data. Parameters allow for control for fine-tuning of the match. This is primarily used as the workhorse for the `tier_match` function.

Usage

```r
merge_plus(
  data1,
  data2,
  by = NULL,
  by.x = NULL,
  by.y = NULL,
  suffixes = c("_1", "_2"),
  check_merge = TRUE,
  unique_key_1,
  unique_key_2,
  match_type = "exact",
  fuzzy_settings = build_fuzzy_settings(),
  score_settings = NULL,
  filter = NULL,
  filter.args = list(),
)```
merge_plus

```r
evaluate = match_evaluate,
evaluate.args = list(),
allow.cartesian = FALSE,
multivar_settings = build_multivar_settings()
```

**Arguments**

- **data1**
  - data.frame. First to-merge dataset (ordering matters - see Fuzzy Matching vignette.)

- **data2**
  - data.frame. Second to-merge dataset.

- **by**
  - character string. Variables to merge on (common across data 1 and data 2). See merge

- **by.x**
  - length-1 character vector. Variable to merge on in data1. See merge

- **by.y**
  - length-1 character vector. Variable to merge on in data2. See merge

- **suffixes**
  - character vector with length==2. Suffix to add to like named variables after the merge. See merge

- **check_merge**
  - logical. Checks that your unique_keys are indeed unique.

- **unique_key_1**
  - character vector. Primary key of data1 that uniquely identifies each row (can be multiple fields)

- **unique_key_2**
  - character vector. Primary key of data2 that uniquely identifies each row (can be multiple fields)

- **match_type**
  - string. If 'exact', match is exact, if 'fuzzy', match is fuzzy. If 'multivar,' match is multivar-based. See multivar_match.

- **fuzzy_settings**
  - additional arguments for amatch, to be used if match_type = 'fuzzy'. Suggested defaults provided. See build_fuzzy_settings.

- **score_settings**
  - list. Score settings for post-hoc matchscores. See build_score_settings

- **filter**
  - function or numeric. Filters a merged data1-data2 dataset. If a function, should take in a data.frame (data1 and data2 merged by name1 and name2) and spit out a trimmed version of the data.frame (fewer rows). Think of this function as applying other conditions to matches, other than a match by name. The first argument of filter should be the data.frame. If numeric, will drop all observations with a matchscore lower than or equal to filter.

- **filter.args**
  - list. Arguments passed to filter, if a function

- **evaluate**
  - Function to evaluate merge_plus output.

- **evaluate.args**
  - list. Arguments passed to evaluate

- **allow.cartesian**
  - whether or not to allow many-many matches, see data.table::merge()

- **multivar_settings**
  - list of settings to go to the multivar match if match_type == 'multivar'. See multivar-match and build_multivar_settings.

**Value**

- list with matches, filtered matches (if applicable), data1 and data2 minus matches, and match evaluation
multivar_match  Matching by computing multivar_scores based on several variables

Description

`multivar_match` computes a multivar_score between each pair of observations between datasets `x` and `y` using several variables, then executes a merge by picking the highest multivar_score pair for each observation in `x`.

Usage

```r
multivar_match(
  data1,
  data2,
  by = NULL,
  by.x = NULL,
  by.y = NULL,
  unique_key_1,
  unique_key_2,
  logit = NULL,
  missing = FALSE,
  wgts = NULL,
  compare_type = "diff",
  blocks = NULL,
  blocks.x = NULL,
  blocks.y = NULL,
  nthread = 1,
  top = 1,
  threshold = NULL,
  suffixes = c("_1", "_2")
)
```

Arguments

data1  data.frame. First to-merge dataset.
data2  data.frame. Second to-merge dataset.
by  character string. Variables to merge on (common across data 1 and data 2). See merge
by.x  character string. Variable to merge on in data1. See merge
by.y  character string. Variable to merge on in data2. See merge
unique_key_1  character vector. Primary key of data1 that uniquely identifies each row (can be multiple fields)
**Description**

Common special characters and their replacements for string cleaning

**Usage**

```r
sp_char_words
```
**Format**

An object of class `data.table` (inherits from `data.frame`) with 4 rows and 2 columns.

<table>
<thead>
<tr>
<th>State_FIPS</th>
<th>State_FIPS</th>
</tr>
</thead>
</table>

**Description**

Data.table with state FIPS codes and abbreviations.

**Usage**

`State_FIPS`

**Format**

An object of class `data.table` (inherits from `data.frame`) with 55 rows and 3 columns.

<table>
<thead>
<tr>
<th>tier_match</th>
<th>Perform an iterative match by tier</th>
</tr>
</thead>
</table>

**Description**

Constructs a `tier_match` by running `merge_plus` with different parameters sequentially on the same data. Allows for sequential removal of observations after each tier.

**Usage**

```r
 tier_match(
  data1,
  data2,
  by = NULL,
  by.x = NULL,
  by.y = NULL,
  suffixes = c("_1", "_2"),
  check_merge = TRUE,
  unique_key_1,
  unique_key_2,
  tiers = list(),
  takeout = "both",
  match_type = "exact",
  clean = FALSE,
  clean_settings = build_clean_settings(),
  score_settings = NULL,
  filter = NULL,
) ```
```r
tier_match =
filter.args = list(),
evaluate = match_evaluate,
evaluate.args = list(),
allow.cartesian = TRUE,
fuzzy_settings = build_fuzzy_settings(),
multivar_settings = build_multivar_settings(),
verbose = FALSE
)
```

**Arguments**

- `data1` *data.frame.* First to-merge dataset.
- `data2` *data.frame.* Second to-merge dataset.
- `by` *character string.* Variables to merge on (common across data 1 and data 2). See `merge`
- `by.x` *character string.* Variable to merge on in data1. See `merge`
- `by.y` *character string.* Variable to merge on in data2. See `merge`
- `suffixes` *see merge*
- `check_merge` *logical.* Checks that your `unique_keys` are indeed unique, and prevents merge from running if merge would result in data.frames larger than 5 million rows
- `unique_key_1` *character vector.* Primary key of data1 that uniquely identifies each row (can be multiple fields)
- `unique_key_2` *character vector.* Primary key of data2 that uniquely identifies each row (can be multiple fields)
- `tiers` *list.* tier is a list of lists, where each list holds the parameters for creating that tier. All arguments to `tier_match` listed after this argument can either be supplied directly to `tier_match`, or indirectly via `tiers`.
- `takeout` *character vector,* either 'data1', 'data2', 'both', or 'neither'. Removes observations after each tier from the selected dataset.
- `match_type` *string.* If 'exact', match is exact, if 'fuzzy', match is fuzzy.
- `clean` *Boolean,* T/F, whether or not to clean strings prior to the match.
- `clean_settings` *list.* Settings for string cleaning. See `clean_strings` and `build_clean_settings`.
- `score_settings` *list.* Settings for post-hoc matchscoring. See `build_score_settings`.
- `filter` *function or numeric.* Filters a merged data1-data2 dataset. If a function, should take in a data.frame (data1 and data2 merged by name1 and name2) and spit out a trimmed version of the data.frame (fewer rows). Think of this function as applying other conditions to matches, other than a match by name. The first argument of filter should be the data.frame. If numeric, will drop all observations with a matchscore lower than or equal to filter.
- `filter.args` *list.* Arguments passed to filter, if a function
- `evaluate` *Function to evaluate merge_plus output.* see `evaluate_match`.
- `evaluate.args` *list.* Arguments passed to function specified by `evaluate`
allow.cartesian
whether or not to allow many-many matches, see data.table::merge()

fuzzy_settings
additional arguments for amatch, to be used if match_type = 'fuzzy'. Suggested
defaults provided. (see amatch, method='jw')

multivar_settings
list of settings to go to the multivar match if match_type == 'multivar'. See
multivar-match.

verbose
boolean, whether or not to print tier names and time to match each tier as the
matching happens.

Details
See the tier match vignette to get a clear understanding of the tier_match syntax.

Value
list with matches, data1 and data2 minus matches, and match evaluation

See Also
merge_plus clean_strings

wgt_jaccard_distance    Computing Weighted Jaccard Distance

Description
#' wgt_jaccard_distance computes the Weighted Jaccard Distance between two strings. It is
vectorized, and accepts only two equal-length string vectors.

Usage
wgt_jaccard_distance(string_1, string_2, corpus, nthreads = 1)

Arguments
string_1    character vector
string_2    character vector
corpus      corpus data.table, constructed with fedmatch::build_corpus
nthreads    number of threads to use in the underlying C++ code

Details
See the vignette fuzzy_matching for details on how the Weighted Jaccard similarity is computed.

Value
numeric vector with the Weighted Jaccard distances for each element of string_1 and string_2.
word_frequency

Compute frequency of words in a corpus

Description

word_frequency counts the frequency of words in a set of strings. Also does minimal cleaning (removes punctuation and extra spaces). Useful for determining what words are common and may need to be replaced or removed with clean_strings.

Usage

word_frequency(string)

Arguments

string character vector

Value

data.table with word frequency

World_Bank_Codes

World Bank 3-Character Country Codes for 213 countries

Description

World Bank 3-Character Country Codes for 213 countries

Usage

World_Bank_Codes

Format

An object of class data.table (inherits from data.frame) with 213 rows and 2 columns.
Index

* datasets
  articles, 2
  corp_data1, 10
  corp_data2, 11
  corporate_words, 10
  fund_words, 11
  sp_char_words, 17
  State_FIPS, 18
  World_Bank_Codes, 21

articles, 2

build_clean_settings, 3
build_corpus, 3
build_fuzzy_settings, 4
build_multivar_settings, 5
build_score_settings, 6
build_tier, 6

calculate_weights, 8
clean_strings, 9
corp_data1, 10
corp_data2, 11
corporate_words, 10

fund_words, 11
fuzzy_match, 12

match_evaluate, 13
merge_plus, 14
multivar_match, 16

sp_char_words, 17
State_FIPS, 18
tier_match, 18

wgt_jaccard_distance, 20
word_frequency, 21
World_Bank_Codes, 21